User's Manual of Flow* v1.2.0

Xin Chen

RWTH Aachen University, Germany.
xin.chen@cs.rwth-aachen.de
November 20, 2013

1 Introduction

Hybrid systems are dynamical systems which exhibit both continuous flow and discrete jumps. They are natural modeling formalism for the systems composed of a discrete controller interacting with a physical environment. Since hybrid systems often appear in safety-critical situations, it is non-trivial to study their behavior. A typical way to do that is to explore the reachable state space of a hybrid system, which is called *reachability analysis*. Unfortunately, such a job can not be done algorithmically, since the reachability problem on hybrid systems is undecidable [1]. The tool Flow* tries to compute an over-approximation for a bounded reachable set of a hybrid system. A reachable set w.r.t. a bounded time horizon $[0, T]$ and a maximum jump depth k is called bounded, it consists of the states each of which can be reached at some time $t \in [0, T]$ via at most k jumps from an initial state of the system. The over-approximation set computed by Flow* is a finite group of Taylor Model (TM) flowpipes. Each flowpipe over-approximates the exact reachable set in a small time interval.

The hybrid system models considered by Flow* are described as follows. The continuous dynamics in a mode should be defined by a non-linear Ordinary Differential Equation (ODE). A mode invariant or a jump guard should be defined by polynomial constraints. A jump reset should be defined by a polynomial mapping. The initial set of a hybrid system can be given by an interval or a TM.

We present a birdview of the tool in Figure 1. Flow* accepts two classes of input files. A *model file* describes a bounded reachability problem on a hybrid or continuous system. The tool tries to compute an over-approximation for the bounded reachable set. A *TM file* includes a state space definition as well as finitely many TM flowpipes. The tool reads those specifications and generates a plotting file for the flowpipes. In both of the cases, an unsafe set can be specified, and Flow* will conservatively check the emptiness of the intersection between flowpipes and the unsafe set.

The output of Flow* consists of the following files. A *plotting file* which provides a 2D visualization of the flowpipes. The format of the plotting file can be determined by users, it can be a Gnuplot file or a Matlab file. A *TM file* which describes the state space of the system and contains all computed TM flowpipes. The file can be reused by Flow*, or the content may also be extracted and analyzed by other tools. If an unsafe set is given and the tool detected some "unsafe" flowpipes, then a counterexample file is generated. The file includes all *potential* unsafe flowpipes as well as the (discrete) computation paths which lead to them.

The two main components of Flow* are the *TM integrator* and the *image computation module*. In the module TM integrator, we provide three different schemes for the integration of non-linear ODEs. The functionalities of them are listed as follows.

- `Poly ODE 1` prefers low-degree polynomial ODEs.
- `Poly ODE 2` prefers high-degree polynomial ODEs.
- `Nonpoly ODE` prefers non-polynomial ODEs.

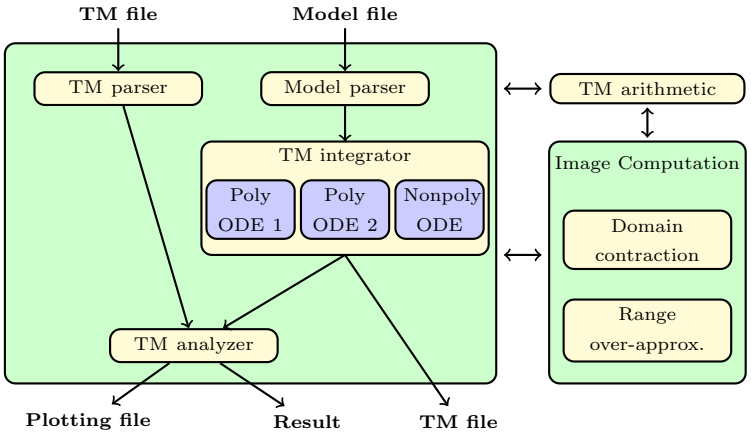


Fig. 1. Structure of Flow*

Although the low-degree polynomial ODEs are also supported by the last scheme, they can be handled more efficiently by the first scheme.

Our tool has a simple installation. After that the libraries and tools listed in Table 1 are properly installed, users only need to type `make` to compile the source code, and then an executable file `flowstar` is generated. Most of the technical details of Flow* are described in [2,3]. Please visit the following web page to fetch the current version of the tool.

`http://ccis.colorado.edu/research/cyberphysical/taylormodels/`

Names	Weblinks
GMP Library	`http://gmplib.org/`
GNU MPFR Library	`http://www.mpfr.org/`
GNU Scientific Library	`http://www.gnu.org/software/gsl/`
GNU Linear Programming Kit	`http://www.gnu.org/software/glpk/`
Bison - GNU parser generator	`http://www.gnu.org/software/bison/`
flex: The Fast Lexical Analyzer	`http://flex.sourceforge.net/`
Gnuplot	`http://www.gnuplot.info/`

Table 1. Libraries and tools used by Flow*

2 A Quick Start

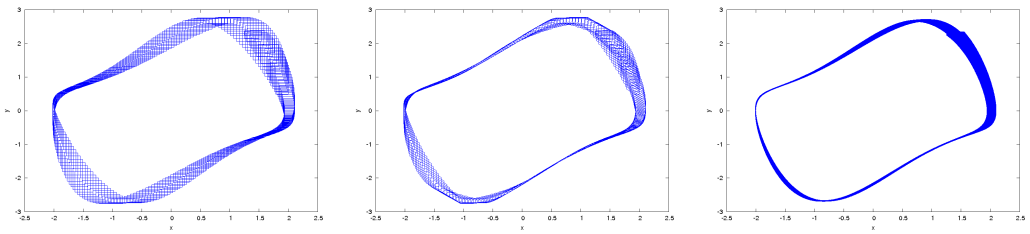
We start the usage of Flow* by a simple example given in [4]. A *Van-der-Pol oscillator* can be modeled by the following ODE:

$$\begin{cases} \dot{x} = y \\ \dot{y} = y - x - x^2 \cdot y \end{cases}$$

We want to compute the TM flowpipes from the initial condition $x \in [1.25, 1.55], y \in [2.25, 2.35]$ in the bounded time horizon $[0, 7]$. Such a continuous reachability task can be described by the following model file.

```
# file vanderpol.model

continuous reachability
{
 state var x,y              # declare the state variables

 setting                    # the reachability setting
 {
  fixed steps 0.02          # time step size
  time 7                    # time horizon
  remainder estimation 1e-5 # [-1e-5,1e-5] in each dimension
  QR precondition           # the preconditioning technique
  gnuplot octagon x,y       # plotting file is in Gnuplot format
  adaptive orders { min {x:4 , y:4} , max {x:6 , y:6} } # TM orders
  cutoff 1e-15              # cutoff threshold
  precision 53              # precision in the computation
  output vanderpol          # name for the output files
  print on                  # display the computation steps
 }

 poly ode 1                 # use the integration scheme Poly ODE 1
 {
  x' =  y
  y' =  y - x - x^2*y
 }

 init                       # specify the initial set
 {
  x in [1.25 , 1.55]
  y in [2.25 , 2.35]
 }
}
```

To make Flow* work on the reachability task, we may simply execute
`$./flowstar < vanderpol.model`
After a while, two files `vanderpol.plt` and `vanderpol.flow` are generated in the subdirectory `outputs`. To visualize the flowpipes, users may use Gnuplot
`$gnuplot < outputs/vanderpol.plt`
and then an `eps` file will be produced in the subdirectory `images`. If the plotting file is specified as a Matlab file, then a `.m` file is generated in `outputs`, and it can be directly executed in the Matlab environment. In Figure 2, we present the three kinds of visualizations of the same TM flowpipes produced by Flow*.



(a) intervals (b) octagons (c) grids

Fig. 2. Different visualizations of the flowpipes

3 Modeling language for continuous reachability problems

We give a detailed introduction for the modeling language of continuous reachability problems in Flow*. To describe a continuous reachability task, we should first define the initial value problem and then specify the parameters which are used in the reachability computation.

An *initial value problem* is defined in the following form in Flow*,

```
poly ode 1
{
 # ODE
}
init
{
 # initial set
}
```

The option `poly ode 1` tells the tool to use the integration scheme `Poly ODE 1` and it can be replaced by one of the other two, i.e., `poly ode 2` and `nonpoly ode`. A d-dimensional ODE can be given by d equations of the form `x' =` φ such that `x` is a state variable and φ can be a polynomial (in the first two integration schemes) or a non-polynomial expression (in the last integration scheme) over the state variables. If φ is a polynomial then it should be defined according to the following syntax

$$\varphi \texttt{ ::= } \varphi \texttt{ + } \varphi \texttt{ | } \varphi \texttt{ - } \varphi \texttt{ | } \varphi \texttt{ * } \varphi \texttt{ | -}\varphi \texttt{ | (}\varphi\texttt{) | }\varphi\texttt{\^{}n | x | r}$$

wherein `n` is a positive integer, `x` is a variable and `r` is a real number. The priorities of the operators are given in Table 2. If φ is a non-polynomial expression, then the corresponding syntax is given as follows

$$\begin{aligned}\varphi \texttt{ ::= } &\varphi \texttt{ + } \varphi \texttt{ | } \varphi \texttt{ - } \varphi \texttt{ | } \varphi \texttt{ * } \varphi \texttt{ | -}\varphi \texttt{ | (}\varphi\texttt{) | }\varphi\texttt{\^{}n | x | r} \\ &\texttt{| } \sin(\varphi) \texttt{ | } \cos(\varphi) \texttt{ | } \exp(\varphi) \texttt{ | } \varphi \texttt{ / } \varphi \texttt{ | sqrt(}\varphi\texttt{)}\end{aligned}$$

in which $\sin, \cos$ are the trigonometric functions sine and cosine respectively, exp is the exponential function, `/` is the division operator and `sqrt` denotes the square root. Flow* also allows users to specify *time-varying disturbances* which are bounded by intervals in ODEs. For example,

```
x' = 1 + 0.5*x*y - x^3 + [-0.01,0.01]
```

The initial set can be an interval or a TM. An interval initial set is given by specifying each variable a single interval, for example

```
x in [1,2]      y in [-1,1]
```

The definition of a TM initial set consists of three parts: (1) TM variable declaration, (2) the TM, and (3) the TM domain. An example is given as below,

```
tm var x0 , x1 , x2
x = 1 + x1^2 - x2 + [-0.02,0.01]
y = x0^3 - x1 + [0,0.1]
x0 in [-0.2,0.2]
x1 in [-0.2,0.2]
x2 in [-0.1,0.1]
```

wherein `x,y` are the state variables.

The parameters in the reachability setting are explained as below.

operators	priorities
^	4
- (prefix)	3
*	2
+	1
- (infix)	1

Table 2. Priorities of the operators

Time step size - The size of TM integration steps. A fixed step size 0.02 can be specified by
`fixed steps 0.02`
To apply an adaptive step size ranges from 0.01 to 0.05, users may write
`adaptive steps { min 0.01, max 0.05 }`

Time horizon - A time horizon $[0, T]$ is given by `time T`.

Remainder estimation - The remainder estimation interval used in TM integration. Users may simply write, for example
`remainder estimation 1e-5`
to tell Flow* to use [-1e-5,1e-5] as the estimation interval in all dimensions. It is also possible to specify different remainder estimations over the dimensions. For example, the following setting tells Flow* to apply [-2e-3,2e-3] to the x dimension and [-0.1,0.1] to the y dimension.
`remainder estimation { x:[-2e-3,2e-3], y:[-0.1,0.1] }`
Small estimations help in control the wrapping effect in flowpipe construction.

Preconditioning technique - Flow* provides two preconditioning techniques to control the wrapping effect. Users may specify `QR precondition` to use the QR preconditioning technique, or use `identity precondition` to apply the identity preconditioning technique. For the systems with more than 3 variables, QR preconditioning technique is *not* suggested to use. More details can be found in [5].

Plot settings - It is difficult to plot even a 2D projection of a TM. Given two state variables x, y, Flow* plots the interval (box) or octagon over-approximations of the flowpipes in the x-y plane. To generate an interval visualization, users may specify
`gnuplot interval x,y` or `matlab interval x,y`
for the Gnuplot output or Matlab output respectively. For octagon visualization, users may just replace the term `interval` by `octagon` in the above specification. Besides, Flow* also provides more accurate visualizations which is based on grid pavings over TM flowpipes. For example, users may use the plot setting `gnuplot grid 10 x,y` to tell Flow* to divide a TM flowpipe into 10^n parts, wherein n is the number of the state variables, by splitting the original domain uniformly into 10 grids in each dimension. Then the plotting is based on the interval enclosures of the subdivisions.

TM orders - The orders of the TMs in flowpipe construction. Theoretically, Flow* may always achieve a better accuracy with higher TM orders. A fixed order 5 in all dimensions can be specified as
`fixed orders 5`
or the user may want to apply a uniform adaptive order ranges from 3 to 6,
`adaptive orders { min 3 , max 6 }`
It is quite common that a system may have several variables which change at very different rates. For such systems, Flow* allows users to specify different fixed or adaptive orders over the state dimensions. For example,
`fixed orders { x:3 , y:4 }` or
`adaptive orders { min { x:2 , y:3 } , max{ x:5 , y:4 } }`

Cutoff threshold - In order to improve the performance, FLOW* simplifies a TM by sweeping the "small" terms in the polynomial part into the remainder interval. For example, the TM

$$[0.5, 0.5] \cdot x + [-1e-5, -1e-5] \cdot y^2 + [0.1, 0.1] \cdot x \cdot y^2 + [5e-4, 5e-4] \cdot x^5 + [-1e-3, 1e-3]$$

over the domain $x, y \in [-1, 1]$ will be simplified to

$$[0.5, 0.5] \cdot x + [0.1, 0.1] \cdot x \cdot y^2 + [1.51e-3, 1.51e-3]$$

if the terms within $[-1e-3, 1e-3]$ are considered "small". Such a threshold can be specified by `cutoff` ε, and then a term which is contained in $[-\varepsilon, \varepsilon]$ will be moved to the remainder.

Precision - the precision of the floating-point numbers in the MPFR library.

Output name - the name of the output files. If `output model` is specified, then the Gnuplot and MATLAB plotting files will be named by `model.plt` and `model.m` respectively, the TM file will be of the name `model.flow`.

Print out - The display of the computation steps can be turned on or off by specifying `print on` or `print off` respectively.

Additionally, an unsafe set can be optionally given after the reachability problem, such as

```
unsafe set
{
 # polynomial constraints
 p1 <= c1
 p2 >= c2
 p3 in [c3,c4]
 ...
}
```

The set should be defined by finitely many polynomial constraints. That is, the set is composed by the states that satisfy all of the constraints. A polynomial constraint should be given as the form `p` $\sim$ `c` or `p in [c1,c2]` wherein `p` is a polynomial over the state variables, `c,c1,c2` are constants and $\sim \in \{<=, >=, =\}$. After the flowpipe construction, FLOW* checks the emptiness of the intersection between the flowpipes and the unsafe set. If it is empty, then the tool reports `SAFE`, otherwise it reports `UNKNOWN` and dumps the potential unsafe flowpipes in a counterexample file the format of which will be described in Section 5.

For the sake of efficiency, the tool uses naive interval arithmetic to check the safety. Therefore, it is better to give *linear constraints* in a reachability problem.

4 Modeling language for hybrid reachability problems

We introduce the modeling language for hybrid reachability problems by a concrete example. The model of a collision avoidance maneuver of two aircraft at a fixed altitude is given in [6]. In the beginning, both of the aircraft are in straight flight with a relative heading. When the aircraft come too close, that is, the distance of them is blow a specified value, the controller will make an instantaneous heading change of 90° on both of them, and then the two aircraft will complete a π-time semicircular arc flying. Afterwards, both aircraft make another 90° instantaneous heading change and resume their headings in the very beginning. Such a system can be modeled by a 3-mode hybrid system, see Figure 3. The state variables

are listed as below,

x_1, y_1 : the coordinates of the first aircraft
α_1 : the heading angle of the first aircraft
x_2, y_2 : the coordinates of the second aircraft
α_2 : the heading angle of the second aircraft
z : timer for the semicircular arc flying

A heading angle is the angle between the heading direction and the vector $(1, 0)^T$ in the coordinate map.

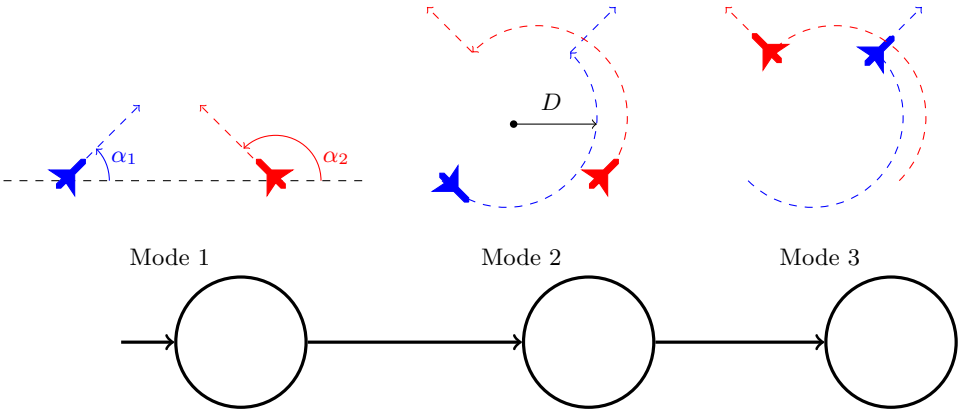


Fig. 3. Hybrid automaton of the collision avoidance maneuver

The continuous dynamics in each mode are of the form

$$\dot{x}_1 = v_1 \cdot \cos\alpha_1,\ \dot{y}_1 = v_1 \cdot \sin\alpha_1,\ \dot{x}_2 = v_2 \cdot \cos\alpha_2,\ \dot{y}_2 = v_2 \cdot \sin\alpha_2$$

The angles α_1, α_2 only changes in Mode 2, and so is the timer z. For the jump from Mode 1 to Mode 2, the guard is given by the constraint $(x_1 - x_2)^2 + (y_1 - y_2)^2 \leq D^2$, and the reset mapping is

$$\alpha_1' := \alpha_1 - \frac{\pi}{2},\ \alpha_2' := \alpha_2 - \frac{\pi}{2}$$

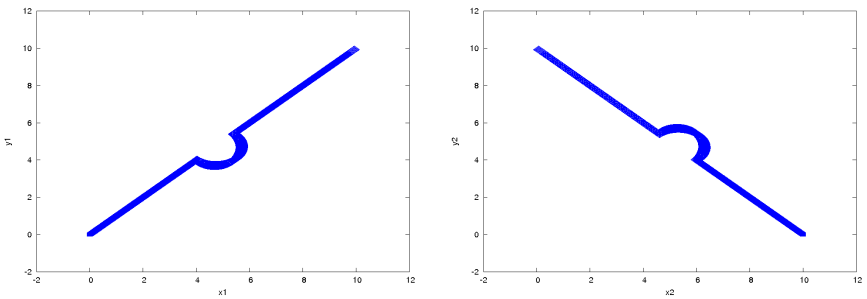
For the jump from Mode 2 to Mode 3, the guard is $z = \pi$, and the reset mapping is same as the above one. Our model file is shown as below wherein we simply choose $v_1 = v_2 = 1$ and $D = 2$.

```
1  hybrid reachability
2  {
3   state var x1, y1, alpha1, x2, y2, alpha2, z
4
5   setting
6   {
7    fixed steps 0.02
8    time 15
9    remainder estimation 1e-6
10   identity precondition
11   gnuplot octagon x1,y1
12   adaptive orders { min 2 , max 5 }
13   cutoff 1e-12
14   precision 53
```

```
 output aircraft
 max jumps 2    # bound of the jump depth
 print on
}

modes
{
 mode1
 {
  nonpoly ode
  {
   x1' = cos(alpha1)  y1' = sin(alpha1)  alpha1' = 0
   x2' = cos(alpha2)  y2' = sin(alpha2)  alpha2' = 0
   z' = 0
  }
  inv { (x1 - x2)^2 + (y1 - y2)^2 >= 4 }
 }

 mode2
 {
  nonpoly ode
  {
   x1' = cos(alpha1)  y1' = sin(alpha1)  alpha1' = 1
   x2' = cos(alpha2)  y2' = sin(alpha2)  alpha2' = 1
   z' = 1
  }
  inv { z <= 3.1415 }
 }

 mode3
 {
  nonpoly ode
  {
   x1' = cos(alpha1)  y1' = sin(alpha1)  alpha1' = 0
   x2' = cos(alpha2)  y2' = sin(alpha2)  alpha2' = 0
   z' = 0
  }
  inv { }
 }
}

jumps
{
 mode1 - > mode2
 guard { (x1 - x2)^2 + (y1 - y2)^2 = 4 }
 reset { alpha1' := alpha1 - 1.57075  alpha2' := alpha2 - 1.57075 }
 parallelotope aggregation { [x1:1,y1:1] [x2:1,y2:1] }

 mode2 - > mode3
 guard { z = 3.1415 }
 reset { alpha1' := alpha1 - 1.57075  alpha2' := alpha2 - 1.57075 }
 parallelotope aggregation { }
}

init
{
 mode1
```

```
72    {
73     x1 in [-0.1,0.1]    y1 in [-0.1,0.1]   alpha1 in [0.785374,0.785376]
74     x2 in [9.9,10.1]    y2 in [-0.1,0.1]   alpha2 in [2.35612,2.35613]
75     z in [0,0]
76    }
77   }
78  }
79
80  unsafe set
81  {
82   mode1 { x1 - x2 <= 0.2  x1 - x2 >= -0.2  y1 - y2 <= 0.2  y1 - y2 >= -0.2 }
83   mode2 { x1 - x2 <= 0.2  x1 - x2 >= -0.2  y1 - y2 <= 0.2  y1 - y2 >= -0.2 }
84   mode3 { x1 - x2 <= 0.2  x1 - x2 >= -0.2  y1 - y2 <= 0.2  y1 - y2 >= -0.2 }
85  }
```

The reachability task considers all the possible initial positions in the region $[-0.1, 0.1] \times [-0.1, 0.1]$ for the first aircraft, and the initial positions in the region $[9.9, 10.1] \times [-0.1, 0.1]$ for the second aircraft. The initial heading angle of the first aircraft is $\frac{\pi}{4}$ and that of the second aircraft is $\frac{3}{4}\pi$. There are at most two jumps in the hybrid automaton, so we bound the jump depth by 2. We set the time horizon $[0, 15]$ such that the two jumps will definitely be executed in 15 time units. The octagon enclosures of the two aircraft trajectories are presented in Figure 4. FLOW* verifies that the two aircraft never got too close to each other.



(a) First aircraft (b) Second aircraft

Fig. 4. Collision avoidance maneuver

Similar to the modeling file of a continuous reachability problem, the description of a hybrid reachability problem is also composed by two parts, i.e., the setting for reachability computation and the initial value problem on a hybrid system. The locations (modes) and jumps of a hybrid system are declared in `modes { ... }` and `jumps { ... }` respectively. A mode is defined by the form

```
name          # name of the mode
{
 poly ode 1   # can also be 'poly ode 2' or 'nonpoly ode'
 {
  # ODE in the mode
 }
 inv
 {
  # polynomial constraints
 }
}
```

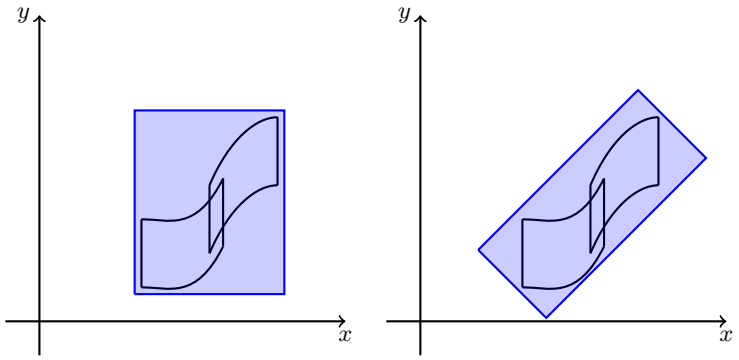
and a jump from mode `m1` to mode `m2` is defined by the form

```
m1 -> m2
guard
{
 # polynomial constraints
}
reset
{
 # polynomial reset mapping with uncertainties
}
... # aggregation setting
```

Flow* accepts polynomial reset mappings given in the form of `x' := p` wherein `x` is a state variable and `p` is a polynomial over the state variables. Additionally, users may also include an interval uncertainty, that is a reset mapping may also be of the form `x' := p + [a,b]`. An unspecified variable will be associated with an identity mapping, that is `x' := x`.

The flowpipe/guard intersection of a jump is usually a non-convex or even a set of disjoint fragments. To avoid dealing with each of them in the consequent computation, we over-approximate (or aggregate) the intersection sets by an order 1 TM (with zero remainder interval) which can be viewed as an parallelotope. It is not hard to compute the transformations between the TM and parallelotope representations, however, to determine a proper orientation for the aggregation set is not easy.

For a d-dimensional parallelotope, its orientation can be determined by d linearly independent vectors[1], the set of which is called *template*. We give an example in Figure 5. The two flowpipes can be over-approximated by a parallelotope with the template $\{(1,0)^T,(0,1)^T\}$, or the template $\{(1,1)^T,(1,-1)^T\}$. It is obvious that the template plays an important role in the aggregation accuracy. In Flow*, we provide the following options to select a template for aggregating the flowpipe/guard intersections for a jump.



(a) template $\{(1,0)^T,(0,1)^T\}$ (b) template $\{(1,1)^T,(1,-1)^T\}$

Fig. 5. Example of flowpipe aggregation

– *Interval aggregation.* Flow* computes a box enclosure for the flowpipe union, similar to the example shown in Figure 5(a). The specification is
`interval aggregation`

[1] They are also the linear independent facet normals of the parallelotope.

– *General parallelotopic aggregation.* Users may simply specify
`parallelotope aggregation { }`
to tell FLOW* to automatically determine the template based on the invariants, guard, flow directions and so on. It is also allowed to give some vectors which will be definitely selected by the tool. For a 3-dimensional example, the following specification
`parallelotope aggregation { [x:1,y:1] [z:2] }`
tells the tool to choose $(1, 1, 0)^T$ and $(0, 0, 2)^T$ in determining the template, the remaining ones will still be selected automatically. Since the vector lengths make no sense in the template, FLOW* always normalizes the vectors to the length 1.

To the best of our knowledge, there is no approach to find the best template for an aggregation job. Hence we will add more heuristics into FLOW* in the future.

5 Formats of the output files

5.1 TM files

For a continuous reachability problem, the TM file generated by FLOW* is of the following format,

```
state var x, y, ...     # state variable declaration
gnuplot octagon x , y   # plot setting
unsafe set
{
 ...                    # defined by polynomial constraints
}
output model            # output name
continuous flowpipes
{
 tm var x0, y0, ...     # declare the variables used in TMs
 ...                    # TM flowpipes
}
```

A TM flowpipe is represented as

```
{
 x = p + [a,b]          # TM in every dimension
 ...

 local_t in [0,r]       # domain of the TM
 x0      in [c,d]
 ...
}
```

wherein `local_t` is the local time variable of the TM. It ranges in a time step interval. Notice that the TM variables are automatically named by the tool, thereby their names have nothing to do with the state variable names.

For a hybrid reachability problem, the generated TM file is of the following form. The flowpipes are kept according to the mode sequences visited in the computation.

```
state var x, y, ...     # state variable declaration

mode1                   # specify the mode invariants
{
 ...
}
```

```
...

computation paths      # computation paths of modes
{
 m1 (n1 , [a1,b1]) -> m2 (n2 , [a2,b2]) -> ... -> mk;
 ...
}

gnuplot octagon x , y  # plot setting
output model           # output name

unsafe set
{
 ...
}

hybrid flowpipes
{
 mode1
 {
  tm var x0, y0, ...    # declare the variables used in TMs
  ...                   # TM flowpipes
 }

 ...                    # other modes in the computation tree
}
```

The computation paths of modes are also dumped. As an example, the path

```
m1 (n1 , [a1,b1]) -> m2 (n2 , [a2,b2]) -> ... -> mk
```

denotes a mode sequence from `m1` to `mk` in the reachability computation. A segment

```
mi (n , [a,b]) -> mj
```

means that there is a jump which is numbered by `n` from the mode `mi` to the mode `mj` such that the execution time is included by the interval `[a,b]`. For a hybrid system, the jumps are numbered by $1, 2, \ldots$ in the order of their declarations.

5.2 Counterexample files

The tool Flow* also provides the possibility to dump all potential unsafe flowpipes. Since the TM flowpipes are over-approximations of the exact reachable set segments, we are not able to prove the unsafety by detecting a nonempty intersection between the TM flowpipes and the unsafe set. However, we may dump those potential unsafe TM flowpipes and they can be used in further analysis work.

For a continuous reachability problem, the unsafe TM flowpipes are dumped into a counterexample file of the following form.

```
{
 starting time t     # starting time of the flowpipe
 {
  ...                # TM flowpipe
 }

 ...
}
```

For a hybrid reachability problem, the counterexample file is as follows,

```
mk                  # mode name
{
 starting time t    # local starting time of the flowpipe
 {
  ...               # TM flowpipe
 }

 ...

 computation path: # the computation path leads to the unsafe flowpipes
 m1 (n1 , [a1,b1]) -> m2 (n2 , [a2,b2]) -> ... -> mk;
}

...
```

Since we compute over-approximations, the counterexamples generated might be superfluous.

6 More case studies

We collected a considerable number of continuous and hybrid system benchmarks in the case study homepage of Flow*. It can be found as below.

`http://systems.cs.colorado.edu/research/cyberphysical/taylormodels/casestudies/`

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.
2. X. Chen, E. Ábrahám, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Proc. of the 33rd IEEE Real-Time Systems Symposium (RTSS'12)*, pages 183–192. IEEE, 2012.
3. X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of the 25th International Conference on Computer Aided Verification (CAV'13)*, LNCS. Springer, 2013.
4. Matthias Althoff. *Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars*. PhD thesis, Technischen Universität München, 2010.
5. K. Makino and M. Berz. Suppression of the wrapping effect by taylor model-based verified integrators: Long-term stabilization by preconditioning. *International Journal of Differential Equations and Applications*, 10(4):353 – 384, 2005.
6. I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *Proc. of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *LNCS*, pages 310–323. Springer, 2000.